ASHRAE - Great Energy Predictor III

Group Name: Big Brain Machine Learners

Group Members & zID:

Michael Agius: z5257081

Ji Hye Kim: z5422963

Oscar Moses: z5160715

Mina Na: z5375535

Lauren Wu: z5259189

1. Introduction

https://www.kaggle.com/competitions/ashrae-energy-prediction

Improvements are being made to buildings across the world in hopes of reducing overall costs and energy emissions. However, to determine whether these improvements are indeed working, we must deduce comparisons between the energy emissions using various meters without any additional manufactured components. The more accurate predictions we make of the building energy usage, the more likely large scale investors will finance in this area to enable the progress of future efficient energy-usage buildings.

To achieve this, we must create a model that accurately predicts the building energy usage in the following areas; chilled water, electric, hot water, and steam meters. We will use the data consisting of over 1000 buildings, collected over a three year timeframe.

We approached this by first garnering an understanding of our data; recognising correlations and distributions, feature relevance, dealing with missing data points, and identifying the relationships between the features. After cleaning our data, we used this to build different experimental training models, before deciding on one that would predict the results as accurately as possible.

2. Exploratory Data Analysis

1. Raw data analysis

The raw data consists of three separate datasets: the target dataset, weather metadata and building metadata. The target dataset contains the values of the target [*meter_reading*] along with supporting features [*building_id*, *meter*, *timestamp*]. The weather metadata provides numerical measures related to weather based on the *site_id*. Hereafter, *Data* refers to the joint dataset created by combining the target dataset, building metadata, and weather metadata into a single flattened dataset.

The data is in tabular form, initially consisting of 15 features and one target. 11 features are numerical and continuous whilst the remaining 4 are nominal, categorical measures. The target variable is a continuous, numeric prediction; how many kilowatt hours (*kWh*) are read at the time of the sample.

A given sample is of the form:

Features: *(meter, timestamp, building_id, site_id, primary_use, square_feet, year_built, floor_count, air_temperature[°C], cloud_coverage[%], dew_temperature[°C], precip_depth_1_hr[mm], sea_level_pressure[mb/hPa], wind_direction[°], wind_speed[m/s])*

Target: *(Meter_reading [kWh])*

The target represents energy usage in four categories; *electricity, chilled water, steam* and *hot water*. For *electricity*, our focus, mean is 158.76 whilst the standard deviation is 368.39 with the distribution heavily leaning towards 0, representing the times of the year in which energy usage is low due to inactivity. There are also high value outliers that resemble a measurement error. [see *Appendix. Graph 2.1.i - Meter Reading Distribution (Log scale)*]

Some weather features, such as sea level pressure and air temperature have a normal distribution across all sites as seen below. In contrast, most of the features do not resemble any common distribution. [see *Appendix. Graph 2.1.ii - sea_level_pressure and air_temperature Distribution]*

In essence, the goal is to predict the kWh value given the building and weather data at each time of the sample.

2. Missing data

a. Missing value ratio per feature

The chart below illustrates *Appendix Graph 2.2.a - The proportion of empty values present per feature*. The results indicate that the *floor_count* feature has a missing value ratio of over 80%, while the *year_built* feature has a missing value ratio of approximately 60%. Given that replacing these missing values with standard approaches, such as replacing *NaN* with *0* or imputing with median or mean values, is not appropriate, it is necessary to exclude these features from the analysis. As a result, the *floor_count* and *year_built* features were dropped from the dataset.

b. Filling in Missing data

Graph 2.2.a indicates that certain features, such as *cloud_coverage, precip_depth_1_hr,* and *wind_direction*, have high empty value ratios. [see *Appendix Graph 2.2.a - The proportion of empty values per feature]* As these features correspond to weather measurements, it was determined that the missing data could be approximated using the nearest available data. Missing rows were filled using the mean or equivalent ratio based on the *timestamp* and *site_id*.

3. Data transformation

This section encompasses the process of data transformation, aimed at improving the representation of each feature in a machine learning model. The transformation encompasses various techniques like data restructuring, which reorganises data according to a selected categorical value. Data conversion is utilised to unify the unit of measure across features and convert timestamp objects to numerical values. Feature scaling is applied to standardise the data, resulting in a zero mean and unit variance, enabling more effective model training.

a. Data restructuring

The *'meter'* feature is nominal, discrete and represents the different meter types. While usage is correlated across all four types of meters, predicting variably typed meter readings with the same input data may be difficult for a single, simple model. We decided to focus on type 0 - electricity - to simplify the initial exercise of constructing a model. Upon *Table 2.3.a.i*, it was observed that the electricity values exhibit a strong representation of the underlying tendencies of the other categorical values in the dataset. [see *Appendix Table 2.3.a.i - Correlation between Meter categorical values*] Furthermore, the quantity of data for electricity is greatest. [see Appendix *Graph 2.3.a.ii - Observation ratio per Meter*]

b. Data Conversion

This section involves unifying the measurement unit in a feature and transforming the timestamp data type to numeric values. This process enables the data to be suitable for use by a machine learning algorithm.

i. Unifying measurement units

The data was recorded across multiple locations, and during the data collection process an error was made at the first site. Meter readings for site 0 are in *kBTU* rather than *kWh*, however this is resolved by simply multiplying each element by a constant conversion factor.

ii. Timestamp numeric transformation

The timestamp measure was split into two features; an integer that represents the time passed since the first date of recording and the hour value alone, providing easy separation between night and day as well as peak and shoulder energy consumption periods. Energy consumption is generally cyclical over a 24 hour period given a period of the year [see Appendix Graph 2.4.i - meter_reading average throughout the day].

iii. Integer Encoding

The 16 categorical values in *primary_use* were assigned to an integer between 0 and 15. This helped to reduce the computational costs of training, and potentially improved performance in identifying patterns in data.

c. Feature Scaling

Feature scaling including Normalisation, Standardisation is for preventing large scale features from weighing more than others. Regularisation prevents overfitting by adjusting weights. These techniques are useful for distance-based regression models, such as linear regression. However, as our model (*3. Methodology*) is partition-based, the relationship between features and the target should not be affected, regardless of whether the features are scaled or not.

d. One-hot Encoding

The other nominal feature was the primary use of the building, ranging across 16 categories like education and residential. The less popular categories were grouped into an “other” category, leaving only 6 afterwards. Initially, we planned to use 1-hot encoding to represent these values, but after deciding on decision tree regression as a model, it proved more effective to encode them as integers where 1 might represent education and 2 for offices etc.

4. Correlation analysis

a. Correlation between meter type

Table 2.3.a.i shows correlation between each category of meter type based on each meter row mean by day with *NaN* rows removed. [see *Appendix Table 2.3.a.i - Correlation between Meter categorical values*]

b. Correlation between weather values

A relationship exists between the features *dew_temperature*, *air_temperature* and *sea_level_pressure*. *dew_temperature* is calculated from these other two values according to the following equation Td = T - ((100 - RH)/5), where Td is dew point, T is *air_temperature* and RH is relative humidity, which can be calculated from the *sea_level_pressure*. [see *Appendix Table 2.3.b.i - Correlation between weather features and target value*]

Wind_speed and *wind_direction* can also have an effect on temperature. Faster *wind_speed* can lower the air pressure and thus lower the apparent *air_temperature*. Additionally, depending on the building location, wind could be a hot or cold wind. For instance, a building in Sydney may experience a cold southerly wind, or a hot westerly wind.

Changes in *air_temperature*, air pressure and wind can change the *cloud_coverage*.

5. Potential Limitation

The circular data format is the best way to interpret numerical values like *wind_direction* and *hour.* For example, because *wind_direction* is measured in degrees, the distance between 20° and 350° in a circular data format is shorter than the distance between 240° and 350°. It is unclear whether dropping these columns is more beneficial due to this reason, or whether keeping them is worthwhile despite their potential to confuse the model. This issue will be re-evaluated through experiments during the model-fitting process.

3. Methodology

1. Methods Developed

The solution was approached inline with the standard learning problem flow. Training data was passed through to the learning algorithm which became the model after reaching the desired accuracy. External to this, hyper-parameter tuning and overfitting analysis informed our engineering decisions during the learning process.

2. Feature Selection:

In *Table 2.3.b.i,* we were unable to identify any distinct correlation features between weather_metadata and the target that could explain a clear relationship. Therefore, we conducted experiments using different feature combinations to select the most relevant features. Since the goal of this table is to identify relevant features from weather_metadata, *square_feet*, and *primary_use* (which are selected from building_metadata), these features were applied as defaults. *Table 3.2.i* displays the R-squared and MSLE for different feature combinations, when fitted with a Decision Tree Regressor model with *max_depth*=30. *Air_temperature*, *cloud_coverage*, *dew_temperature*, and *precip_depth_1_hr* show high compatibility in terms of |*correlation*| and are used as the base combination. We added *sea_level_pressure* and *wind_speed*, which show high accuracy and low MSLE, to the base pair. [see *Appendix Table 3.2.i - R-squared and MSLE with different weather feature combinations*]

The *Table 3.2.ii* compares the accuracy for evaluating the assumption made in *2.5 Potential Limitation*. [see *Appendix Table 3.2.ii - R-squared and MSLE with feature assumed to have potential limitation*]

As stated in section *2.2.a*, the dataset no longer includes *floor_count* and *year_built*.

Dew_temperature, *precip_depth_1_hr*, *sea_level_pressure*, and *hour* show the highest accuracy and reasonable correlation. Therefore, these features, along with *square_feet* and *primary_use* selected from building_metadata, were used. It's worth noting that the *hour* was parsed from the *timestamp* and the *timestamp* itself was also used.

3. Hyper-Parameter Tuning

Two methods were used for hyper-parameter tuning; grid search and quantitative observation. Whilst grid searching was effective, it was time consuming and brute force in nature. Observing the quantitative results, especially visually, yielded an important discovery - that the *tree_size* and *depth* could be substantially reduced whilst still maintaining a high level accuracy. [see Appendix *Graph 3.3.i - Decision tree accuracy over max_depth on training and test dataset*]

When grid searching, the model_selection.GridSearchCV from Scikit Learn's library was used to compare the accuracy of various configurations of parameters. The process was conducted on a random sample of the training set (66% of all train-data) with 5-fold cross validation. Even the best combination of non-default hyperparameters yielded marginally worse results than the default. The parameters tested included the split criterion (i.e. *squared_error* or *Poisson*), max tree depth(*max_depth*), minimum number of samples per leaf(*min_sample_leaf*), maximum number of features to consider when splitting(*max_features*) and maximum number of leaf nodes(*max_leaf_nodes*).

Visually observing accuracy changes over hyper-parameter spaces showed that trees of substantially smaller depth had only marginally lower performance on the test set. Without any depth constraint, the decision tree regressor had a depth between 60 and 73 and an accuracy between 93% - 97% depending on train/test split ratio. When constrained to a depth of only 25, the tree still had an accuracy of 93%. In situations where model size and performance are paramount, tree depth can be reduced.

4. Evaluation Metric

While experimenting with the model, we initially used a Regression Score to predict the accuracy of the target variable "*meter_reading*", which consists of continuous variables. This was a viable evaluation metric to use for this particular model, as the model tackled a regression problem predicting quantitative data using *R-squared* score. This particular score determines the accuracy of our model using the distance and residual results, between the predicted and actual values, and in our model's case produced an accuracy between 0.91 to 0.97 depending on the randomised proportion taken from the data set.

However, this evaluation metric is not necessarily the most informative if you want to measure the average error distance between the predicted and actual value. As a result, we attempted to use other evaluation metrics such as RMSE to provide more information into the magnitude of these errors, and how they affect the overall accuracy of the model.

Root Mean-Squared Error (RMSE) measures the average magnitude of the overall error, by taking the square root of the mean-squared difference between the predicted and actual value. Unlike Mean Absolute Error, RMSE does not take the absolute value of the difference. As a result, RMSE is informative when determining the direction of the error, and penalises outliers and large errors with heavier weight. RMSE is useful for determining the magnitude of error in its entirety, where large errors are considered more harshly. When applying RMSE over our model however, we receive a significantly high value of 106.6, implying a large number of errors/outliers that are heavily impacting our model.

Because of this, we looked into the possibility of using Root Mean-Squared Logarithmic Error, as this evaluation caters better to variables with a wider range of values, and does not penalise this as harshly as RMSE would. The difference between RMSLE and RMSE is that RMSLE calculates the RMSE of the logarithm of the predicted and actual result, meaning there is a bigger emphasis on the differences between small values, rather than values with larger differences. Without penalising the nature of our data set, it focuses more appropriately on the accuracy of our model instead. Our model also did not require normalisation or regularisation, making RMSLE a suitable evaluation metric.

4. Results

We experimented and implemented two different models to compare the output and results of each. One of these models involved Linear Regression with one-hot encoding, and our final model used a Decision Tree Regressor.

1. Model Selection

We initially chose a random sample set using 10% of the data, and applied Regularisation on each of the data columns before centering the target. We then split the raw data into training and test sets, and feature vs target columns, then applied a linear regression model over it. The linear regression model achieved poor accuracy, generally reaching 34-40%.

As such, our analysis revealed a poor performance of the linear regression model, suggesting that the assumed linear relationship between the input and output variables was false. In order to capture the non-linearities in the data, we instead opted for a Decision Tree Regression model. The decision tree regression model proved to be a powerful tool for capturing complex relationships between variables in our data set, achieving over 97% accuracy. It demonstrated superior handling of both continuous and categorical input variables without requiring any transformation of categorical data, such as one-hot encoding.

2. Train-Validation Ratio

Train-validation split ratios of 0.4, 0.33, 0.3, 0.25, 0.2 and 0.1 were tested with no hyperparameters. A ratio of 0.1 yielded the lowest test set RMSLE of ~0.32. [see *Appendix Table 4.2.i - RMSLE for Various Train-Validation Ratios without hyperparameter tuning*] Further experiments were run by parameterizing the *max_leaf_nodes* hyperparameter [see *Appendix Table 4.2.ii - RMSLE for Various Train-Validation Ratios with max_leaf_nodes tuning*] and the *max_depth* hyperparameter. [see *Appendix Table 4.2.iii - RMSLE for Various Train-Validation Ratios with max_depth tuning*]

3. Hyperparameters

Through hyperparameter tuning, we experimented with using no hyperparameters, as well as parameterising *max_leaf_nodes* [see *Graph 4.3.i - RMSLE over log(max_leaf_nodes) on train and test data* and *Graph 4.3.ii - log(tree size) over log(max_leaf_nodes) on train and test data*] and *max_depth* [see *Graph 4.3.iii - RMSLE over max_depth on train and test data* and *Graph 4.3.iv - log(tree size) over max_depth on train and test data*]. We noticed from a tree leaf node count of 3,000,000 and onwards, training error decreases whilst test error increases implying that the model was overfitting the data from that point forward. In contrast, when parameterising *max_depth*, the point of overfitting occurs at depth 30. [see *Appendix Table 4.2.iii*] Although both methods yielded a similar test error with only a slight difference of 0.0004 in RMSLE, they resulted in different tree sizes. Considering that the *max_leaf_nodes* approach projected a 36.33%(1 - (5999999 / 9423771) reduction in the tree size compared to the alternative method, as shown in *Appendix Table 4.2.ii* and *Appendix Table 4.2.iii* , we decided to modify the *max_leaf_nodes* hyperparameter for the final model. A *max_leaf_nodes* value of 3,000,000 allowed for a smaller, simpler model, providing better performance in terms of time and space complexity.

5. Discussion

1. Insight of selected model and data

In a decision tree regression model, overfitting occurs when the model becomes too complex and begins to fit the training data too closely. This can result in poor generalisation to new, unseen data. One of the main causes of overfitting in decision tree models is having too many leaf nodes or a tree that is too deep. In order to avoid overfitting, it is important to select appropriate hyperparameters such as *max_leaf_nodes* and *max_depth*. The experiments conducted to vary these hyperparameters allowed us to gain insights into the trade-offs between model complexity and accuracy. By identifying the optimal hyperparameter values, we improved the model’s performance and ensured that it generalised well to new data.

Our initial model used a simplistic linear regression model, and handled missing data by dropping these columns without any obvious removal of specific outliers that we had observed. Although this can be more inclusive of all the data collected, this might result in severe loss of information, and can also perform poorly because the outliers in the data set haven’t been catered for. This was observed in this linear regression model’s final accuracy score rated at 34%. From these results, we knew to opt for another model that better suited the nature of the data as the relationships between the features were non-linear, and decided with the Decision Tree Regressor as it was also much easier to interpret the data.

For our final model, it handles missing data by dropping the data matching *meter_type* = 0. This allows a heavier focus on the model’s prediction accuracy, rather than penalising it due to uncorrelated data that would impact the model. Because our chosen model also consists of a Decision Tree Regressor, when dealing with outliers it is relatively robust. However, to reduce the possibility for large impact of outliers on the model, we removed some empty columns before having trained the model.

2. Limitations of Model

The initial linear regression model faced limitations in that the model always assumed a linear relationship between the features in the data. The model was also prone to overfitting if the data is too complex, and is more sensitive to overall outliers as a result. Lastly, due to the nature of the linear regression model, there is an assumption that all data points collected are independent variables from one another. In cases where the data may occasionally be dependent such as weather, time, etc, the model penalises these instances by providing an estimation of the regression coefficients that is heavily biassed.

On the other hand, limitations of using our decision tree model include the possibility for overfitting, the complexity in handling continuous variables and sensitivity to a wider range of values and noticeable outliers. To curb some of these concerns, the use of our chosen evaluation metric RMSLE helps to limit the model's reliance on such outliers when we attempt to interpret its accuracy. Although this better evaluates the nature of the model’s predictions, there are still limitations in that a Decision Tree Regressor does not work off of weights and is a non-parametric model, and rather just partitions in itself. This still makes the model prone to overfitting, especially given the nature of the dataset’s complexity.

6. Conclusion

Decision tree regression is effective in learning and accurately predicting energy consumption. The *max_depth* and *max_leaf_nodes* hyper-parameters play a significant role in limiting model size and a moderate role in the mitigation of overfitting.

A series of experiments compared various hyperparameter configurations. By selecting appropriate hyperparameters and utilising techniques such as cross-validation, we were able to achieve a Root Mean Squared Logarithmic Error of 0.317364 on our test set. Considering our target variable has a relatively high variance, it demonstrates the model's effectiveness in predicting the target variable.

While the results of our analysis are promising, there are still several areas that could be improved upon if we had more time. For example, we could explore a larger variety of alternate machine learning algorithms, such as the Ensemble Model using bagging, boosting or stacking, to see if they provide better accuracy or less error. In particular, stacking enables the identification of a model that is compatible with the unique tendencies exhibited by each dataset. The model for time series forecasting such as ARIMA can be considered as well, the dataset provided yearly *meter_reading* flow by time. Experimenting with more diverse models would have significantly improved our understanding of the data, and may have enabled us to achieve a model that could compute a higher accuracy.

Additionally, we could work on identifying and addressing outliers in our data since our target variable displays a large range of values. Outliers are data points that differ significantly from other points and can have a major impact on the performance of a model. By detecting outliers, we can potentially improve the performance of our model. These improvements could lead to even better results and a more robust model.

The availability of data and the reasonable predictability of the target is conducive to a strong use case. By leveraging data-driven methods, it is possible to accurately predict energy consumption patterns and optimise energy usage. This has significant implications for both cost saving and reducing energy consumption. The activity of using machine learning algorithms to estimate energy consumption is undoubtedly a fruitful endeavour that can yield tangible benefits for individuals and organisations.

References

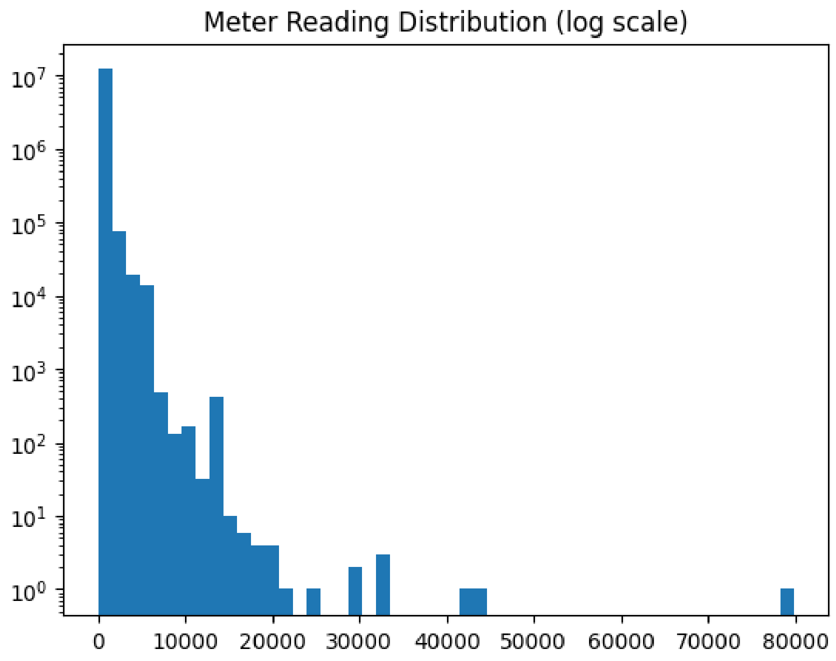
Wesner, J. (2016) *Mae and RMSE - which metric is better?*, *Medium*. Human in a Machine World. Available at: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d (Accessed: April 3, 2023).

Decision tree regression with hyper parameter tuning (2021) *Decision Tree Regression With Hyper Parameter Tuning In Python*. NBShare Notebooks. Available at: https://www.nbshare.io/notebook/312837011/Decision-Tree-Regression-With-Hyper-Parameter-Tuning-In-Python/ (Accessed: April 3, 2023).

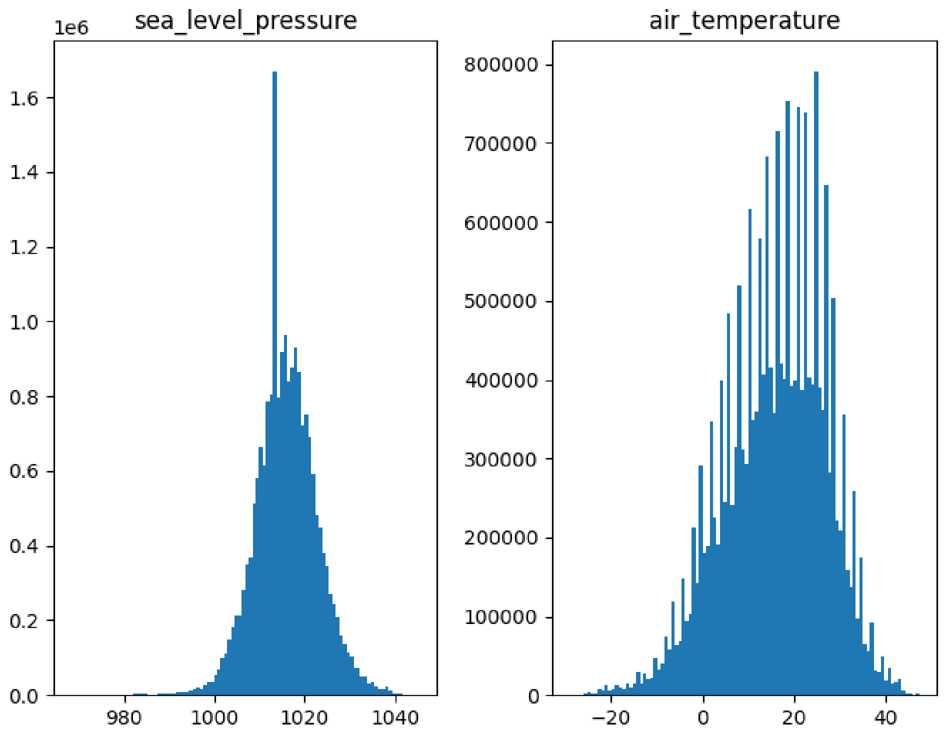
Bruyn, R.de (2021) *Hyper-parameter tuning for scikit-learn ML models*, *Medium*. DVT Software Engineering. Available at: https://medium.com/dvt-engineering/hyper-parameter-tuning-for-scikit-learn-ml-models-860747bc3d72 (Accessed: April 2, 2023).

Appendix

Graph 2.1.i - Type 0 Meter (Electricity) Reading Distribution (Log scale)



Graph 2.1.ii - sea_level_pressure and air_temperature Distribution



Graph 2.2.a - The proportion of empty values present per feature

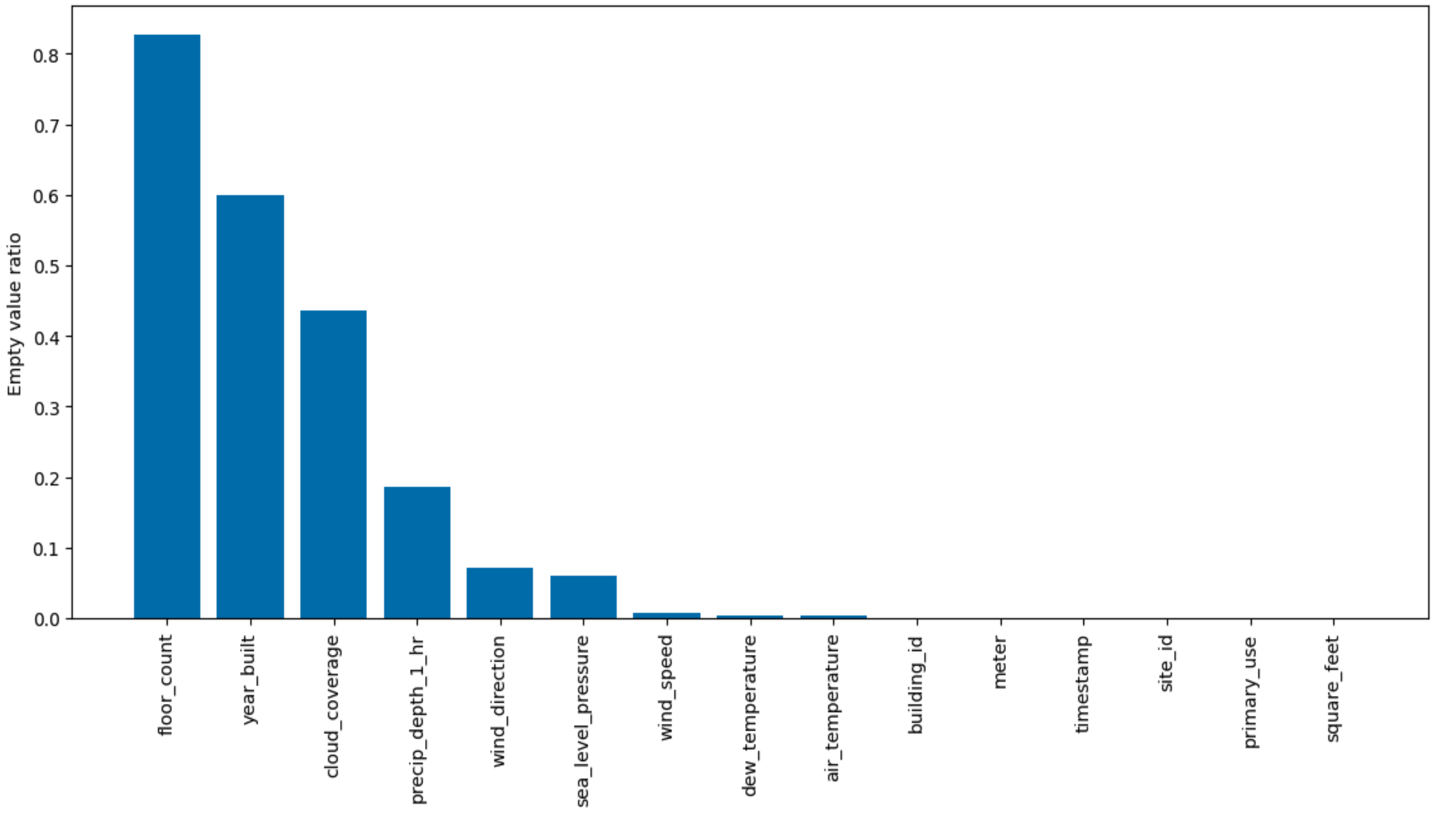


Table 2.3.a.i - Correlation between Meter categorical values

[47]:

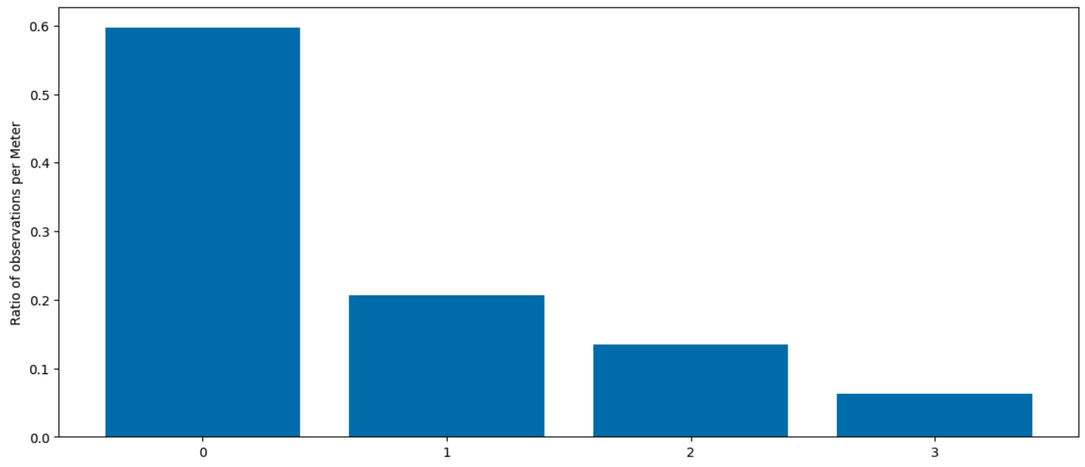
	meter_reading_elec	meter_reading_cw	meter_reading_steam	meter_reading_hw
meter_reading_elec	1.000000	0.691538	0.791354	0.167005
meter_reading_cw	0.691538	1.000000	0.443113	-0.099179
meter_reading_steam	0.791354	0.443113	1.000000	0.292504
meter_reading_hw	0.167005	-0.099179	0.292504	1.000000

Table 2.3.b.i - Correlation between weather features and target value

[245]:

	air_temperature	cloud_coverage	dew_temperature	precip_depth_1_hr	sea_level_pressure	wind_speed	meter_reading
air_temperature	1.000000	0.048662	0.794468	-0.026786	0.121414	0.087572	0.197720
cloud_coverage	0.048662	1.000000	-0.060328	0.206313	-0.071748	0.157966	-0.202546
dew_temperature	0.794468	-0.060328	1.000000	-0.069158	0.161993	0.119524	0.169993
precip_depth_1_hr	-0.026786	0.206313	-0.069158	1.000000	-0.076075	0.011801	0.340361
sea_level_pressure	0.121414	-0.071748	0.161993	-0.076075	1.000000	-0.312851	-0.026176
wind_speed	0.087572	0.157966	0.119524	0.011801	-0.312851	1.000000	-0.034219
meter_reading	0.197720	-0.202546	0.169993	0.340361	-0.026176	-0.034219	1.000000

Graph 2.3.a.ii - Observation ratio per Meter



Graph 2.4.i - meter_reading average throughout the day

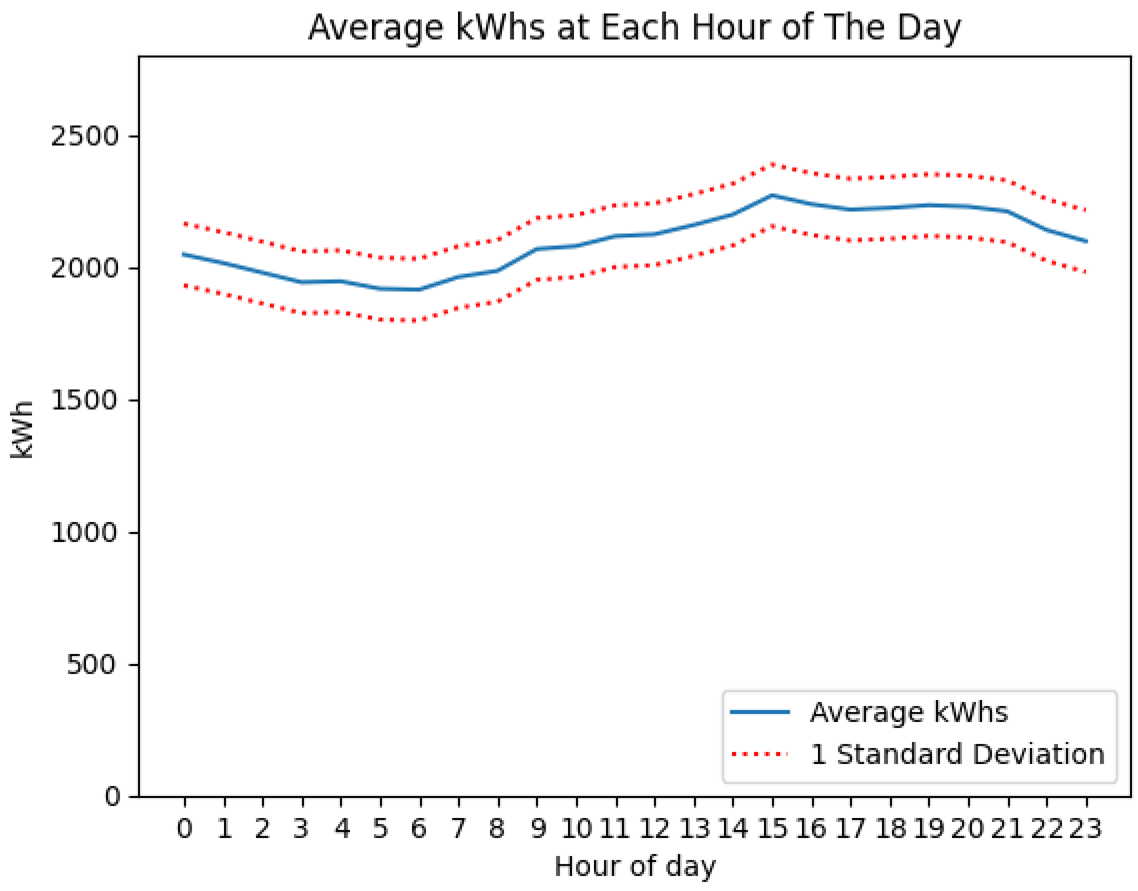
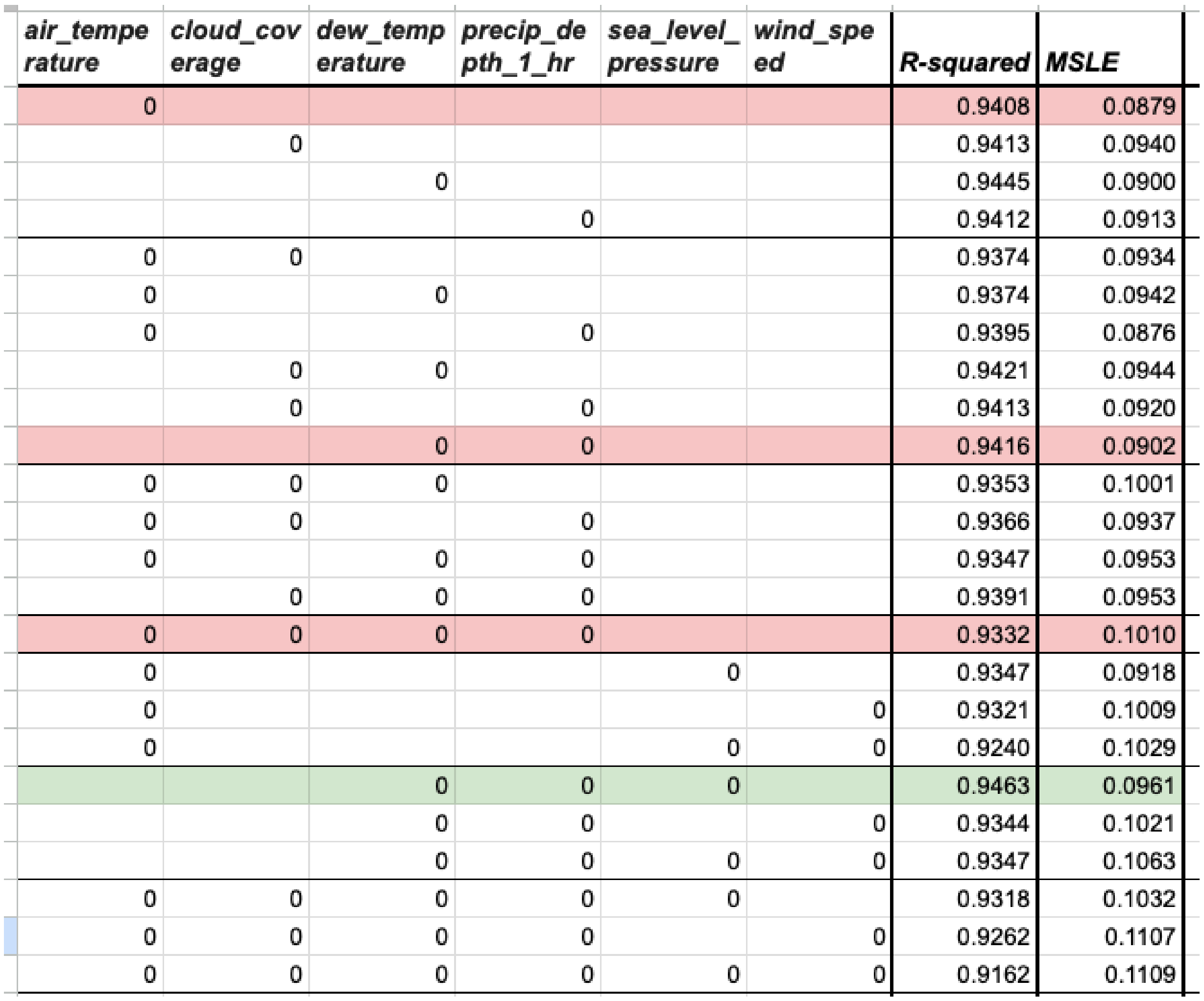
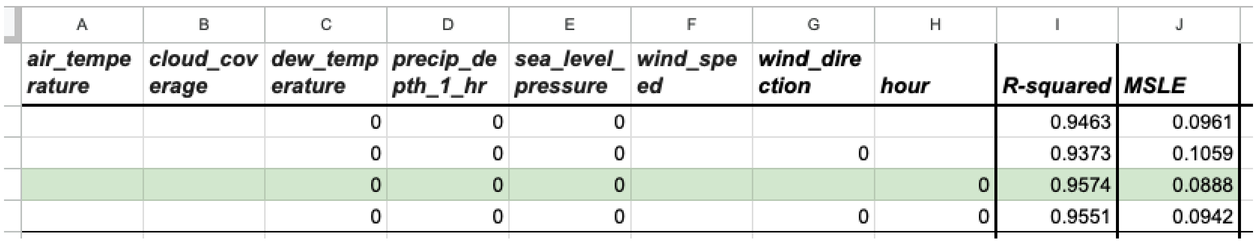


Table 3.2.i - R-squared and MSLE with different weather feature combinations



air_temperature	cloud_coverage	dew_temperature	precip_depth_1_hr	sea_level_pressure	wind_speed	R-squared	MSLE
0						0.9408	0.0879
	0					0.9413	0.0940
		0				0.9445	0.0900
			0			0.9412	0.0913
0	0					0.9374	0.0934
0		0				0.9374	0.0942
0			0			0.9395	0.0876
	0	0				0.9421	0.0944
	0		0			0.9413	0.0920
		0	0			0.9416	0.0902
0	0	0				0.9353	0.1001
0	0		0			0.9366	0.0937
0		0	0			0.9347	0.0953
	0	0	0			0.9391	0.0953
0	0	0	0			0.9332	0.1010
0				0		0.9347	0.0918
0					0	0.9321	0.1009
0				0	0	0.9240	0.1029
		0	0	0		0.9463	0.0961
		0	0		0	0.9344	0.1021
		0	0	0	0	0.9347	0.1063
0	0	0	0	0		0.9318	0.1032
0	0	0	0		0	0.9262	0.1107
0	0	0	0	0	0	0.9162	0.1109

Table 3.2.ii - R-squared and MSLE with feature assumed to have potential limitation



A	B	C	D	E	F	G	H	I	J
air_temperature	cloud_coverage	dew_temperature	precip_depth_1_hr	sea_level_pressure	wind_speed	wind_direction	hour	R-squared	MSLE
		0	0	0				0.9463	0.0961
		0	0	0		0		0.9373	0.1059
		0	0	0			0	0.9574	0.0888
		0	0	0		0	0	0.9551	0.0942

Graph 3.3.i - Decision tree accuracy over max_depth on training and test dataset

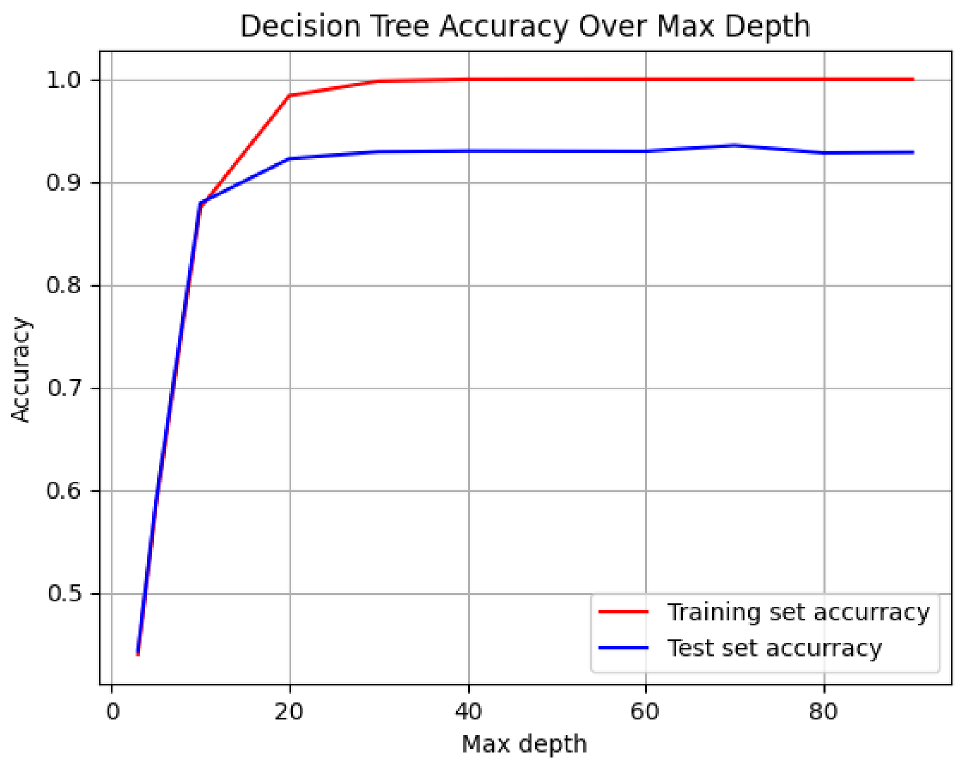
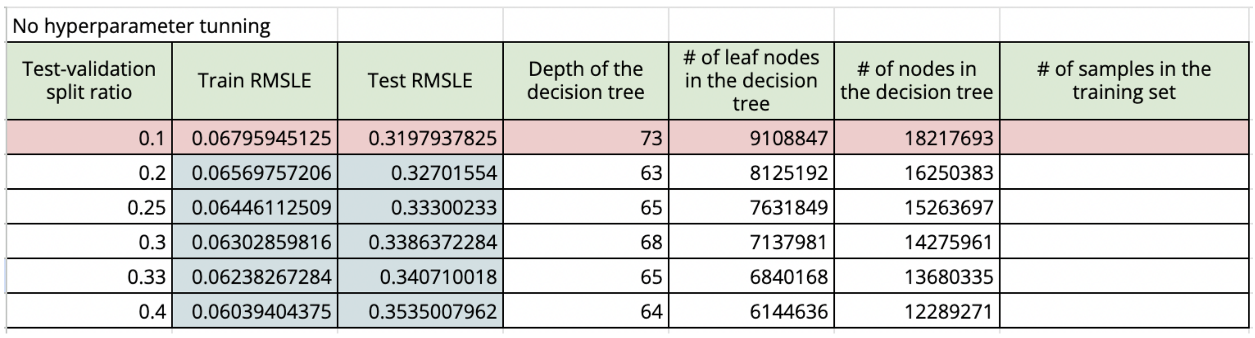


Table 4.2.i - RMSLE for Various Train-Validation Ratios without hyperparameter tuning



No hyperparameter tunning						
Test-validation split ratio	Train RMSLE	Test RMSLE	Depth of the decision tree	# of leaf nodes in the decision tree	# of nodes in the decision tree	# of samples in the training set
0.1	0.06795945125	0.3197937825	73	9108847	18217693	
0.2	0.06569757206	0.32701554	63	8125192	16250383	
0.25	0.06446112509	0.33300233	65	7631849	15263697	
0.3	0.06302859816	0.3386372284	68	7137981	14275961	
0.33	0.06238267284	0.340710018	65	6840168	13680335	
0.4	0.06039404375	0.3535007962	64	6144636	12289271	

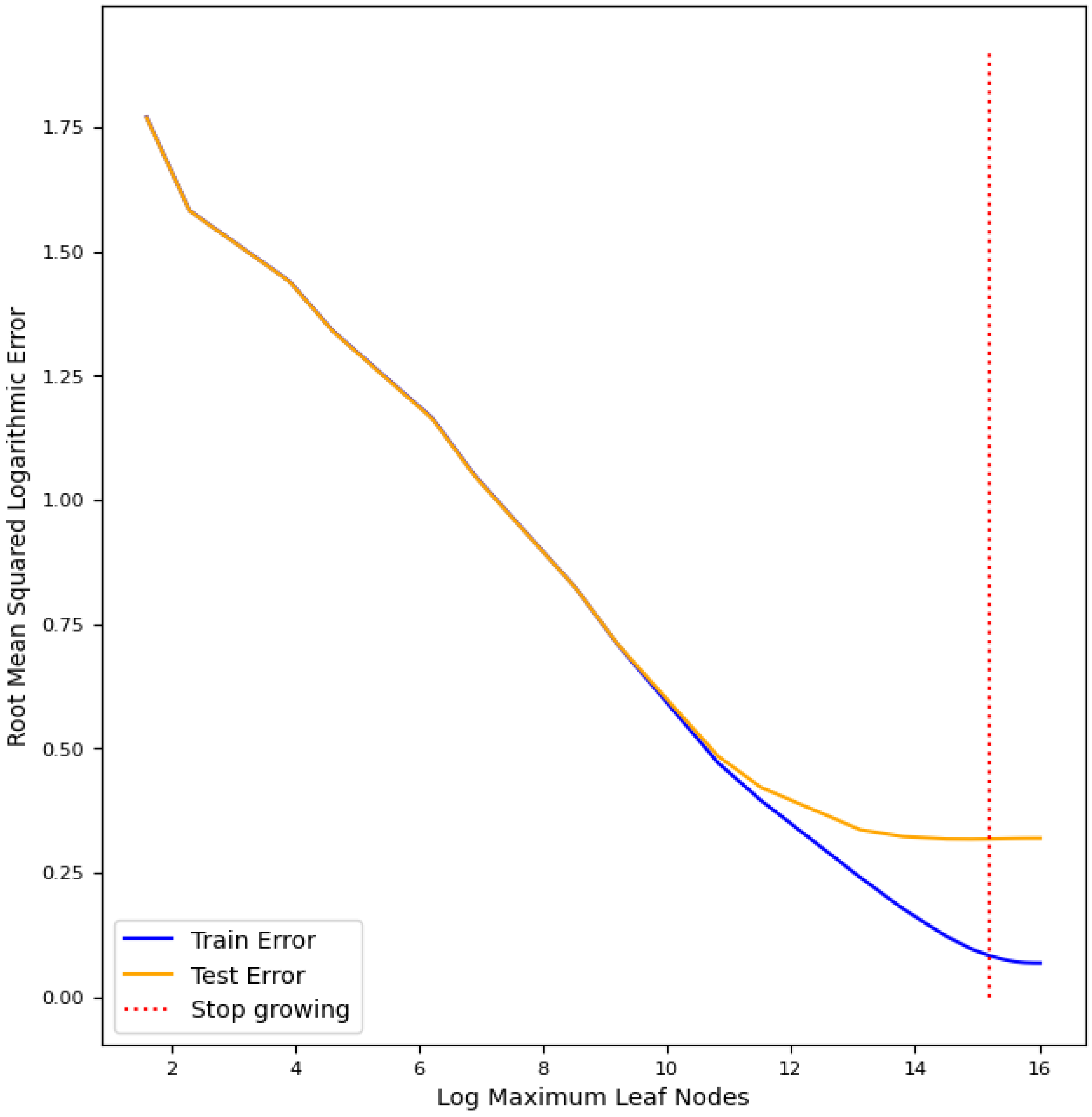
Table 4.2.ii - RMSLE for Various Train-Validation Ratios with *max_leaf_nodes* tuning

Hyperparameter: max_leaf_nodes						
Test-validation split ratio	Train RMSLE	Test RMSLE	Depth of the decision tree	# of leaf nodes in the decision tree	# of nodes in the decision tree	Constraints
0.1	0.06795968403	0.3190040667	73	9000000	17999999	max_leaf_nodes=9000000
0.1	0.06805192537	0.3189924448	73	8125347	16250693	max_leaf_nodes=8125347
0.1	0.06808736491	0.3189879429	73	8000000	15999999	max_leaf_nodes=8000000
0.1	0.06871240705	0.3189134089	72	7000000	13999999	max_leaf_nodes=7000000
0.1	0.07064995412	0.3186899238	72	6000000	11999999	max_leaf_nodes=6000000
0.1	0.07516094161	0.3182494786	69	5000000	9999999	max_leaf_nodes=5000000
0.1	0.08286428003	0.317752159	67	4000000	7999999	max_leaf_nodes=4000000
0.1	0.09624316899	0.3173637906	65	3000000	5999999	max_leaf_nodes=3000000
0.1	0.1215676812	0.3176740159	62	2000000	3999999	max_leaf_nodes=2000000
0.1	0.1764618346	0.3224484937	61	1000000	1999999	max_leaf_nodes=1000000
0.1	0.2399142997	0.3361765258	59	500000	999999	max_leaf_nodes=500000
0.1	0.3956034707	0.4217323071	53	100000	199999	max_leaf_nodes=100000
0.1	0.4710257006	0.4845134776	53	50000	99999	max_leaf_nodes=50000
0.1	0.7076427101	0.7085506371	51	10000	19999	max_leaf_nodes=10000
0.1	0.8243286193	0.8238900767	51	5000	9999	max_leaf_nodes=5000
0.1	1.048251831	1.04689835	25	1000	1999	max_leaf_nodes=1000
0.1	1.16498236	1.163518799	20	500	999	max_leaf_nodes=500
0.1	1.340768416	1.339864135	13	100	199	max_leaf_nodes=100
0.1	1.439998209	1.439257122	11	50	99	max_leaf_nodes=50
0.1	1.581670108	1.581107355	5	10	19	max_leaf_nodes=10
0.1	1.76967598	1.768725596	3	5	9	max_leaf_nodes=5

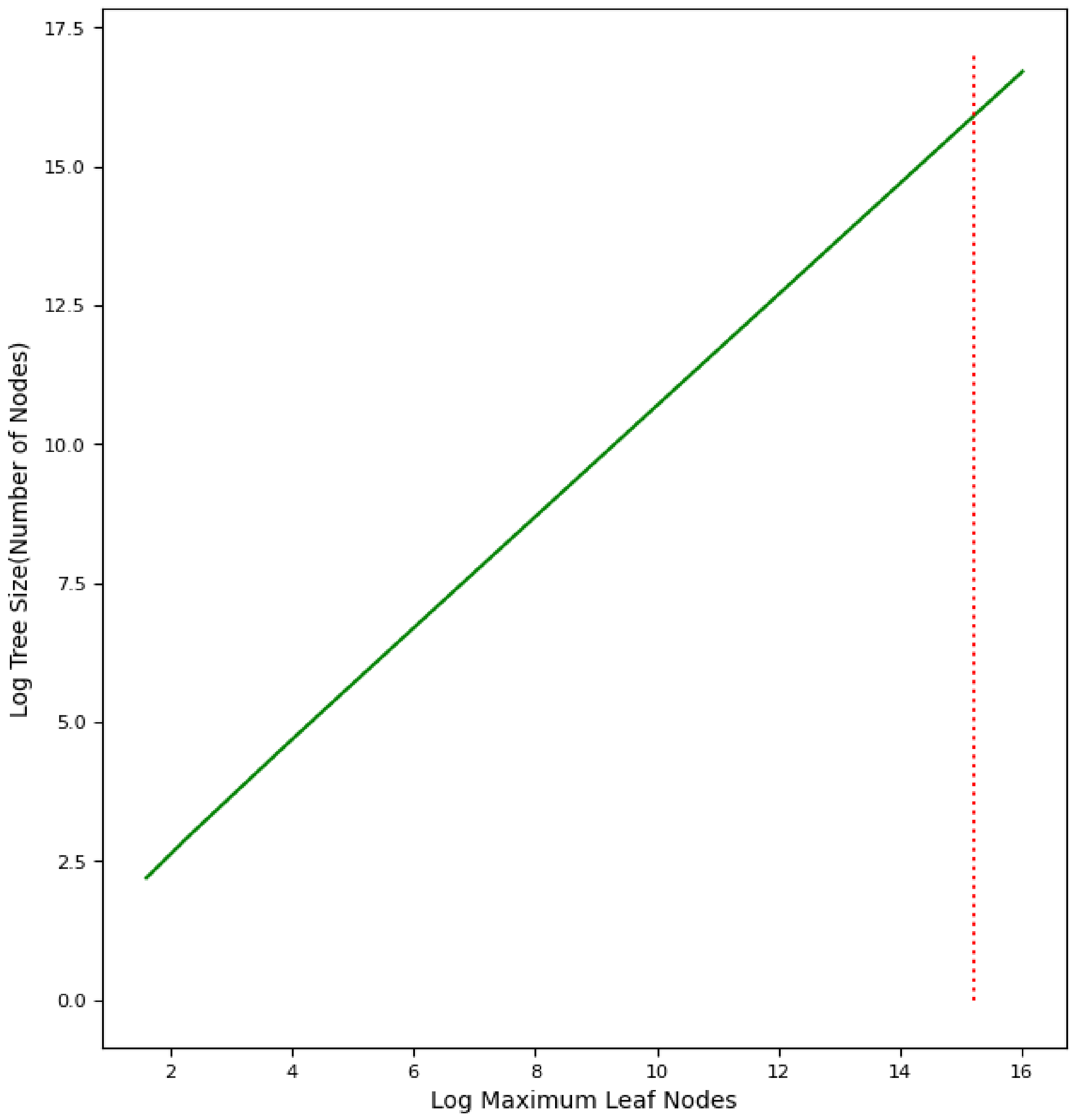
Table 4.2.iii - RMSLE for Various Train-Validation Ratios with *max_depth* tuning

Hyperparameter: max_depth						
Test-validation split ratio	Train RMSLE	Test RMSLE	Depth of the decision tree	# of leaf nodes in the decision tree	# of nodes in the decision tree	Constraints
0.1	0.06795945125	0.3197937825	73	9108847	18217693	max_depth=73
0.1	0.06795945242	0.3197947455	70	9108837	18217673	max_depth=70
0.1	0.06796013776	0.3195339747	60	9108628	18217255	max_depth=60
0.1	0.06809447123	0.3192571681	50	9093733	18187465	max_depth=50
0.1	0.0736760193	0.3199363054	40	8654718	17309435	max_depth=40
0.1	0.1481981889	0.3177653708	30	4711886	9423771	max_depth=30
0.1	0.4724485683	0.4928749767	20	321275	642549	max_depth=20
0.1	1.108958163	1.107609343	10	990	1979	max_depth=10
0.1	1.371766949	1.370619454	5	32	63	max_depth=5

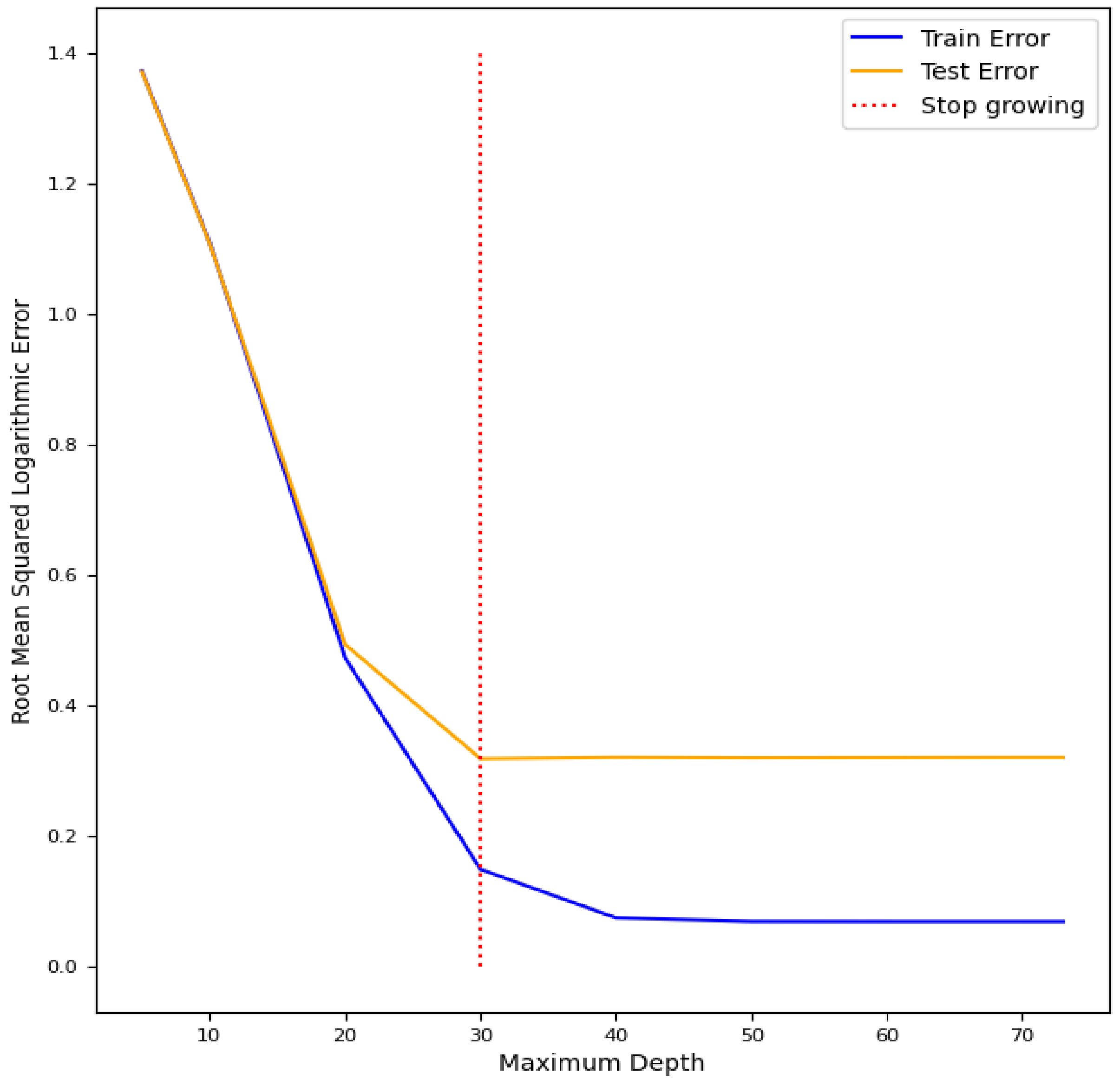
Graph 4.3.i - RMSLE over log(max_leaf_nodes) on train and test data



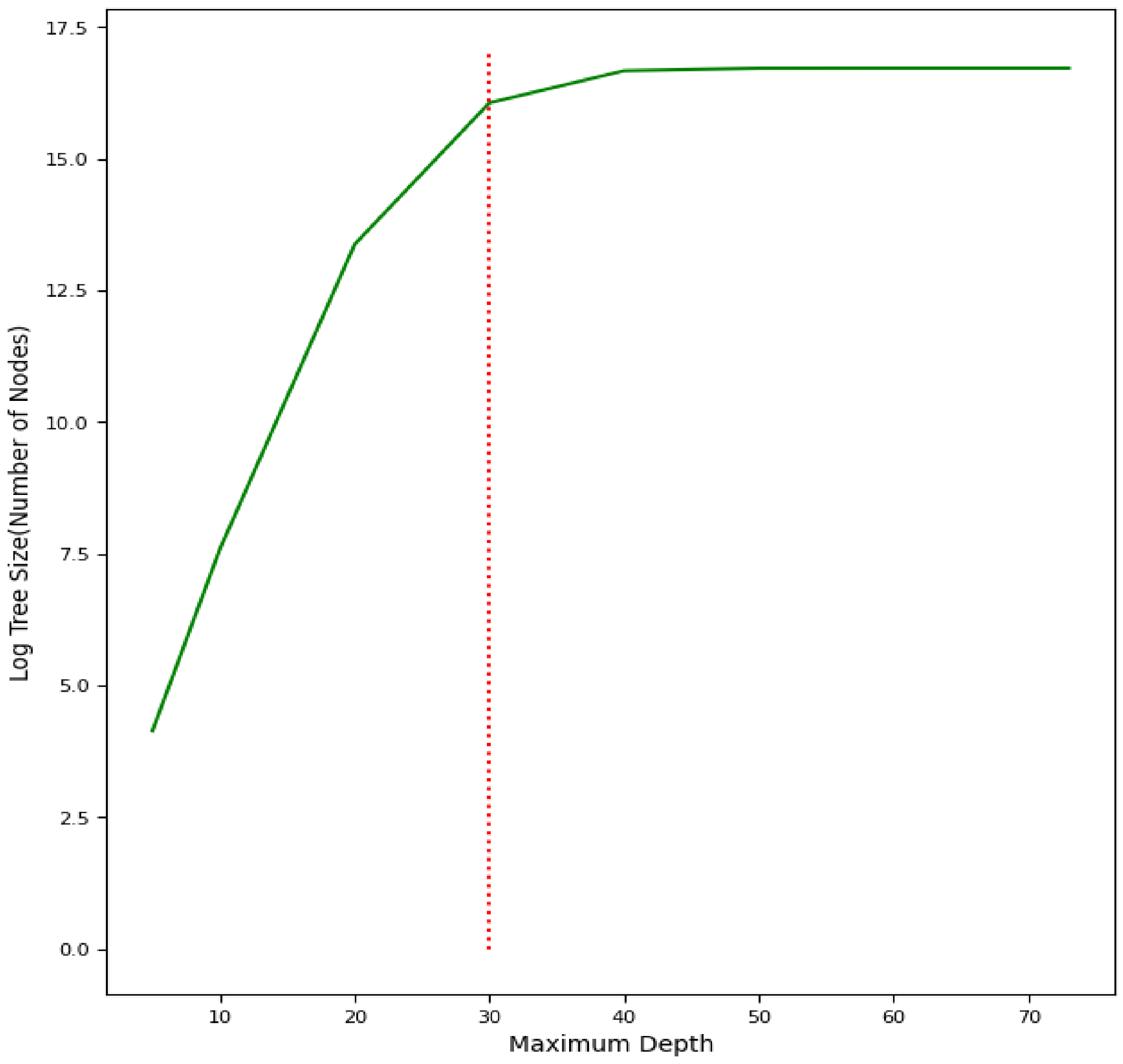
Graph 4.3.ii - log(tree size) over log(max_leaf_nodes) on train and test data



Graph 4.3.iii - RMSLE over max_depth on train and test data



Graph 4.3.iv - log(tree size) over max_depth on train and test data


Log Tree Size(Number of Nodes)
Maximum Depth